재공학을 위한 원시 Software와 대상 Software의 동일성에 관한 연구

김세종○ 이문근
전북대학교 실시간 시스템 연구실
kimsj@chonbuk.ac.kr, moonkun@chonbuk.ac.kr

A Study on Equivalence between Source Software and Target Software in Reengineering

Sejong Kim○ Moonkun Lee
Chonbuk National University Real-Time System Lab.

요 약

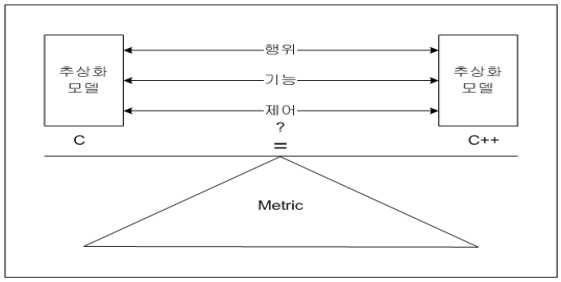
소프트웨어의 유지•보수에 대한 필요성이 커짐에 따라 서로 다른 소프트웨어 사이의 동일성에 대한 검증이 중요한 문제가 되었다. 본 논문에서는 새로운 패러다임을 적용해 변환된 프로그램과 기존의 프로그램 사이의 행위, 기능, 제어 측면의 비교를 통해 두 프로그램 사이의 동일성을 검증하는 방법을 제시한다.

1. 서론

소프트웨어는 제작된 후 지속적인 유지•보수가 필요하다. 개발된 시스템들은 사용자의 새로운 요구, 새로운 기술의 개발, 새로운 환경에 맞도록 수정 보완되어야 하며 이는 막대한 비용을 필요로 한다.[1]

이러한 흐름에 맞추어 시스템을 재공학할 때 유지•보수 및 재사용적 측면에서 기존의 방식들보다 유리한 패러다임을 적용하여 기존의 시스템을 재개발하는 방법들이 제시되었다.[2] 기존의 프로그램과 변환된 프로그램 사이의 일관성을 유지하는 일이 무엇보다 중요해졌음에도 이에 대한 연구는 많이 이루어지지 않고 있다.

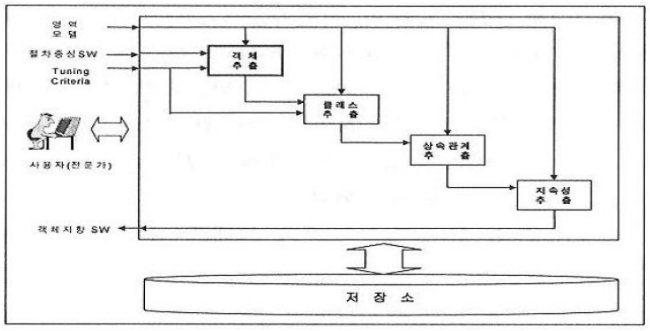
서로 다른 패러다임을 가진 프로그램들의 동일성을 완벽히 검증하는 일이 현실적으로 불가능하기 때문에 각각의 프로그램이 갖는 행위를 추상화시켜 그들이 가지는 의미의 비교를 통해 동일성의 검증을 용이하게 할 수 있다. 본 논문에서는 프로그램 코드의 모델링을 통해 C와 C++언어로 제작된 두 프로그램의 동일성 비교에 대한 방법을 제안한다. 동일성 검증에는 [2]의 방법론을 통해 객체화된 C프로그램과 영역 전무가의 C++프로그램을 사용한다.



<그림 1> 프로그램 모델링을 통한 C언어와 C++언어의 동일성 비교

2. 관련 연구

[2] 에서는 객체를 추출하기 위해 <그림 2> 과 같은 절차를 따라 진행한다. 객체 추출 단계에서는 전역 변수, 사용자 자료형, 함수와 파라미터를 기반으로 관련성 정도를 기준으로 클러스터링하고 클래스 추출 단계에서는 클러스터링된 객체 후보들의 공통적인 특성을 추출하여 클래스를 추출한다. 클래스 추출 단계의 결과는 클래스 간에 대등한 평면 관계를 이루고 있다. 상속관계 추출 단계는 전 단계에서 추출된 평면화된 클래스들의 공통적 특성을 추출하여 *part-of* 관계나 *is-a* 관계를 추출하여 계층 구조를 만든다. 지속성 추출 단계에서는 절차지향 프로그램에는 존재하지 않는 객체지향 특성들과 동적인 부분을 추가한 후 객체지향 소프트웨어를 생성한다.



<그림 2> 절차지향 SW의 객체지향 SW로의 재공학 절차 흐름도

[3]에서는 소프트웨어의 특성을 컨셉이라는 개념으로 세분화하여 동일한 특성을 갖는 컨셉을 묶어 슈퍼 컨셉을 만들었다. 이렇게 만들어진 슈퍼 컨셉과 하위 컨셉들 사이의 관계를 Lattice그래프로 표현하고 서로소인 컨셉들의 집합으로 파티션을 정의하고 이들을 하나의 객체로 정의하여 객체지향의 패러다임에 적용하였다.

3. 동일성 추상화

본 논문에서는 서로 다른 패러다임의 두 언어 (C언어, C++언어)를 모델링하여 그것들이 가지는 행위, 기능, 제어 측면의 비교를 통해 두 프로그램의 동일성을 검증하는 방법을 제시한다.

3.1. 절차지향 SW의 객체지향 SW로의 변환

코드의 비교는 두 소프트웨어의 가장 기본적인 비교 수단이다. 현실적으로 이런 비교 방법은 불가능하기 때문에 소프트웨어의 모델링을 통해 비교를 가능하도록 한다. 하지만 서로 다른 패러다임의 소프트웨어를 비교하는 것 또한 힘들기 때문에 우선적으로 절차중심의 C언어를 객체 중심의 모델로 변환하는 과정이 필요하다. 본 논문에서는 [2]를 통해 객체화된 C언어 모델과 UML을 통해 모델화된 C++언어의 동일성을 비교하기 위해 State Chart, Data Flow, Control Flow 상의 동일성 비교 방법을 제시한다. 이들은 각각 다음과 같이 정의한다.

[정의 1] 소프트웨어 구성 요소

$S = \{O_i \mid 1 \le i \le n\}$

$O_i = \langle O_i^S, O_i^A \rangle$

$O_i^S = \langle S_I, S_1, \ldots, S_n, S_F \rangle$ 여기에서

S_I : 객체의 시작상태.

S_F : 객체의 종료상태를 나타낸다.

$O_i^A = \langle E_A + \rangle$

$E_A = \{E_A \| E_A \mid E_A \triangle E_A \mid (E_A, E_A) \mid A_i\}$

$S_i^T = \langle O_1, O_2, \ldots O_n \rangle$

$O_i^T = \langle A_1, A_2, \ldots, A_n \rangle$ ■

이를 살펴보면, 소프트웨어 S는 $\{O_1, O_2, \ldots, O_n\}$와 같은 객체의 집합으로 구성되어있으며 각각의 객체는 $\langle O_i^S, O_i^A \rangle$의 튜플로 정의한다. 여기서 O_i^S는 객체의 시작상태로부터 종료상태까지의 전이를 순차적으로 표기하는 튜플이며 $\langle S_I, S_1, \ldots, S_n, S_F \rangle$와 같이 쓰인다. 각각의 상태 S_i는 객체를 이루는 함수들이 갖는 반환타입을 표현하며 오버로드, 오버라이드 함수가 있을 수 있으므로 $S_{functionname(returntype)}$와 같이 함수명과 반환 값을 이용해 표기한다. O_i^A는 객체의 시작점으로부터 종료점까지의 처리흐름을 정의하는 것으로 Activity와 Expression으로 구성된다. Expression은 ‖(Synchronization Bar)와 △(판단)으로 나뉘며, Synchronization Bar는 객체 내분의 처리흐름 중 병렬처리절차가 시작하거나 모이는 곳에서 사용되고 판단은 논리식의 결과에 따라 두 곳 이상의 흐름으로 분기가 일어나는 곳에 쓰인다. Activity A_i는 $A_{functionname}$와 같이 표기하고 처리흐름 안에서의 작업이나 행위를 뜻한다. S_i^T와 O_i^T는 전이 순서를 표현하는 튜플로, S_i^T는 소프트웨어 내부의 한 객체에서 다른 객체로의 전이를 순차적으로 표기하고 O_i^T는 객체 내부에서 한 함수에서 다른 함수로의 전이를 순차적으로 표기한다.

3.2. State Chart 동일성

State Chart 다이어그램은 하나의 객체를 대상으로 생존 기간 동안 가질 수 있는 객체 상태의 변화를 분석한 다이어그램이다. 객체 상태와 함께 객체 상태 변화를 유발하는 이벤트와 동작을 정의하는데, 이러한 요소들을 통해 “객체 O는 이벤트 E에 의해 상태 S로 변화하고 그 상태에서 A라는 행위를 한다.”라는 식의 분석이 가능하다. State Chart 다이어그램의 구성요소는 상태(State), 시작상태(Initial State), 종료상태(Final State)와 전이(Transition)으로 나눌 수 있으며 시작상태와 종료상태는 각각 객체의 상태변화가 시작되는 곳과 종료하는 곳을 나타낸다. 상태(State)는 객체가 가질 수 있는 조건이나 상황을 뜻하며 소프트웨어의 관점에서는 객체가 상황에 따라 수행하는 각각의 함수에 대한 결과를 객체의 상태라고 할 수 있다.

두 프로그램의 임의의 객체의 State Chart 동일성은 내부 상태 변화의 포함 여부에 따라 Strong State Chart 동일성과 Weak State Chart 동일성으로 구분한다.

3.2.1 Strong State Chart 동일성

Strong State Chart 동일성은 두 객체가 생존 기간 중 가지는 모든 상태 변화를 비교하여 이것이 모두 동일할 때 Strong State Chart 동일성을 갖는다고 정의한다.

[정의 2] Strong State Chart 동일성 ($\sim_S$)

객체의 시작상태로부터 종료상태까지의 전이를 순차적으로 표현한 튜플

$O_1^S = \langle S_1, S_2, \ldots, S_n \rangle$, $O_2^S = \langle S_1', S_2', \ldots, S_n' \rangle$에 대하여

만약 $S_1 = S_1' \,\&\, S_2 = S_2' \,\&\, \ldots \,\&\, S_n = S_n'$ 임을 만족하면

Strong State Chart 동일성 $O_1^S \sim_S O_2^S$이 성립한다. ■

3.2.2 Weak State Chart 동일성

일반적으로 두 객체가 완전히 동일한 상태의 변화를 갖는 것은 매우 어렵다. 이것은 소프트웨어에 존재하는 함수/프로시져등에 의해 각각은 지역 상태를 가지게 됨에 따라 미묘한 차이를 발생할 수 있기 때문이다.

결론적으로, Weak State Chart 동일성은 객체의 시작상태와 종료상태만의 비교를 통해 두 객체 사이의 동일성을 비교하며 다음과 같이 정의한다.

[정의 3] Weak State Chart 동일성 ($\approx_S$)

객체의 시작상태로부터 종료상태까지의 전이를 순차적으로 표현한 튜플

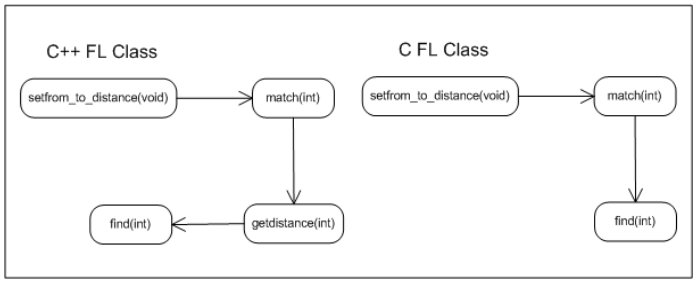
$O_1^S = \langle S_1, S_2, \ldots, S_n \rangle$, $O_2^S = \langle S_1', S_2', \ldots, S_n' \rangle$에 대하여

만약 $S_1 = S_1' \,\&\, S_n = S_n'$ 임을 만족하면

Weak State Chart 동일성 $O_1^S \approx_S O_2^S$이 성립한다. ■

3.2.3 State Chart 동일성 예제

[2]에서 사용된 C언어와 C++언어 프로그램의 FL 클래스를 모델링 해보면 다음과 같은 결과를 얻을 수 있다.



<그림 3> C++언어와 C언어의 State Chart 동일성 비교

<그림 3>의 State Chart를 O_S를 이용해 표현해보면

$FL_{C++} = \langle S_{setfrom\_to\_distance(void)}, S_{match(int)}, S_{getdistance(int)}, S_{find(int)} \rangle$

$FL_C = \langle S_{setfrom\_to\_distance(void)}, S_{match(int)}, S_{find(int)} \rangle$

과 같이 나타낼 수 있다. [정의 2]에 의해 시작상태인 $S_{setfrom_to_distance(void)}$로부터 종료상태 $S_{find(int)}$사이의 모든 상태들이 동일한 과정을 거쳐야 Strong State Chart 동일성이 존재한다고 할 수 있지만 FL_{C++}과 FL_C의 경우에는 시작상태와 종료상태만이 동일하므로 Weak State Chart 동일성만을 만족하고 있다.

3.3. Data Flow 동일성

Data Flow Diagram(DFD) 은 데이터가 소프트웨어 내의 각 프로세스를 따라 흐르면서 변환되는 모습을 나타낸 그림으로 모든 업무를 데이터의 관점에서 접근하여 흐름을 파악한다. 구조적인 방법론에 적용하기 위해 제안되었으나 객체중심의 방법론에도 손쉽게 적용가능하며 프로세스의 세분화/추상화를 통해 로우레벨과 하이레벨의 데이터 흐름을 파악하기 용이하다.

본 논문에서는 객체 집합으로 구성된 두 소프트웨어에 대하여 데이터가 한 객체에서 다른 객체로 전이되는 과정을 비교하여 참조하는 객체의 순서가 동일하다면 두 소프트웨어 사이에 Data Flow 동일성이 존재한다고 정의한다. 또한 객체 내부의 함수/프로시져 사이의 전이 순서를 비교하여 객체들 사이의 Data Flow 동일성을 비교한다.

3.3.1 Object Data Flow 동일성

Object Data Flow 동일성은 소프트웨어가 갖는 객체들 사이의 Data Flow 동일성을 검증하는 것으로, 한 객체에서 다른 객체로의 전이순서를 비교하는 동일성 비교 방법이다.

[정의 4] Obejct Data Flow 동일성 ($\sim_D$)

객체 사이의 전이 조건을 순차적으로 나타내는 튜플

$S_1^T = \langle O_1, O_2, \ldots, O_n \rangle$와

$S_2^T = \langle O_1', O_2', \ldots, O_n' \rangle$에 대하여

만약 $O_1 = O_1'$ & $O_2 = O_2'$ & ... & $O_n = O_n'$ 임을 만족하면

Object Data Flow 동일성 $S_1^T \sim_D S_2^T$이 성립한다. ■

3.3.2 Procedure Data Flow 동일성

Procedure Data Flow 동일성은 소프트웨어를 구성하는 객체 안에서의 Data Flow를 비교하는 방법이다. 함수/프로시져 단위의 전이 순서를 비교하여 함수 사이의 전이 순서가 동일하다면 두 소프트웨어는 Procedure Data Flow 동일성을 만족한다.

[정의 5] Procedure Data Flow 동일성 ($\approx_D$)

함수 사이의 전이를 순차적으로 나타내는 튜플

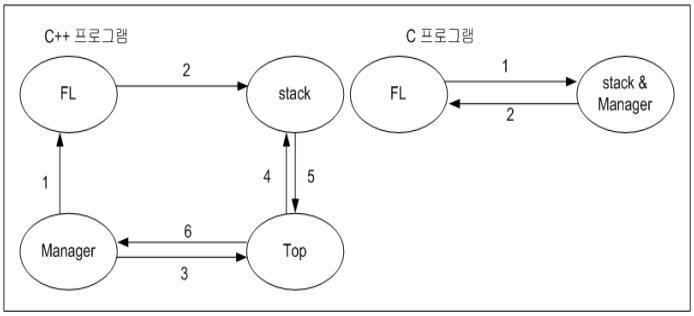
$O_1^T = \langle A_1, A_2, \ldots, A_n \rangle$와

$O_2^T = \langle A_1', A_2', \ldots, A_n' \rangle$에 대하여

만약 $A_1 = A_1'$ & $A_2 = A_2'$ & ... & $A_n = A_n'$임을 만족하면

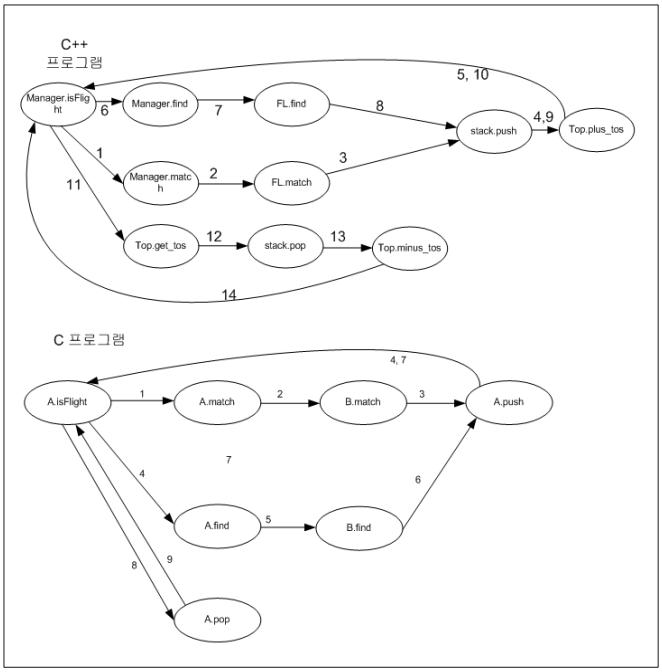
Procedure Data Flow 동일성 $O_1^T \approx_D O_2^T$이 성립한다. ■

3.3.3 Data Flow 동일성 예제



<그림 4> C++언어와 C언어의 Object Data Flow 동일성 비교

예제 프로그램에 대한 객체 사이의 DFD를 그려보면 <그림 4>와 같다. 위에서 볼 수 있듯이 4개의 객체와 2개의 객체 사이의 Data Flow 동일성을 검증하기는 불가능하기 때문에 Object Data Flow 동일성은 판별할 수 없다. 하지만 객체를 세분화하여 함수 사이의 흐름으로 나누어본다면 <그림5>과 같다.



<그림 5> C++언어와 C언어의 Procedure Data Flow 동일성 비교

이 DFD를 O_T의 형식으로 바꾸어 보면

$DF_{C++} = \left\langle \begin{matrix} A_{Manager.isFlight}, A_{Manager.match}, A_{FL.match}, \\ A_{stack.push}, A_{Top.plus\_tos}, A_{Top.get\_tos}, \\ A_{stack.pop}, A_{Top.minus\_tos} \end{matrix} \right\rangle$ 이고

$DF_C = \left\langle \begin{matrix} A_{A.isFlight}, A_{A.match}, A_{B.match}, A_{A.push}, A_{A.isFlight}, \\ A_{A.find}, A_{B.find}, A_{A.push}, A_{A.isFlight}, A_{A.pop} \end{matrix} \right\rangle$

이다. 현 상태에서 보면 둘 사이의 Data Flow는 다르다고 할 수 있지만 DFD의 작성 시 C++프로그램의 Top 클래스를 stack 클래스에 포함시켜 작성할 경우 두 DFD의 흐름이 동일함을 알 수 있다. 결국 두 프로그램은 Procedure Data Flow 동일성을 만족한다.

3.4. Control Flow 동일성

Control Flow 동일성은 UML의 Activity 다이어그램을 이용하여 비교하게 된다. Activity 다이어그램은 업무 영역이나 시스템 영역에서 다양하게 존재하는 각종 처리 로직이나 조건에 따른 처리흐름을 순서에 따라 정의한 모델로서 순차적인 처리 이외에도 병렬적인 처리 또한 제공한다. Activity 다이어그램을 구성하는 요소는 시작점과, 종료점, 액터(Actor), 판단(Decision)과 Synchronization Bar로 나눌 수 있으며 시작점과 종료점은 각각 객체가 처리흐름이 시작하고 종료하는 곳을 의미한다. 액터는 처리흐름 안에서의 작업이나 행위를 뜻하며 판단은 단순히 논리식의 결과에 따라 두 곳 이상의 흐름으로 분기가 일어나는 곳을 말한다. Synchronization Bar의 경우 병렬

처리절차가 시작하거나 모이는 곳을 의미하며 이를 이용해 순차적이면서도 병렬적인 작업을 표시할 수 있다.

Data Flow 동일성이 객체들 사이의 전이가 일어나는 순서에 대한 동일성을 비교한다면 Control Flow 동일성은 객체 내부의 함수/프로시져들이 수행하고 있는 작업에 초점을 맞춰 전이에 따른 Activity들 사이의 처리 흐름에 대한 동일성을 검증한다.

[정의 6] Control Flow 동일성 ($\sim_C$)

객체의 시작점으로부터 종료점까지의 처리흐름을 표현하는 튜플 O_1^A, O_2^A에 대하여

만약

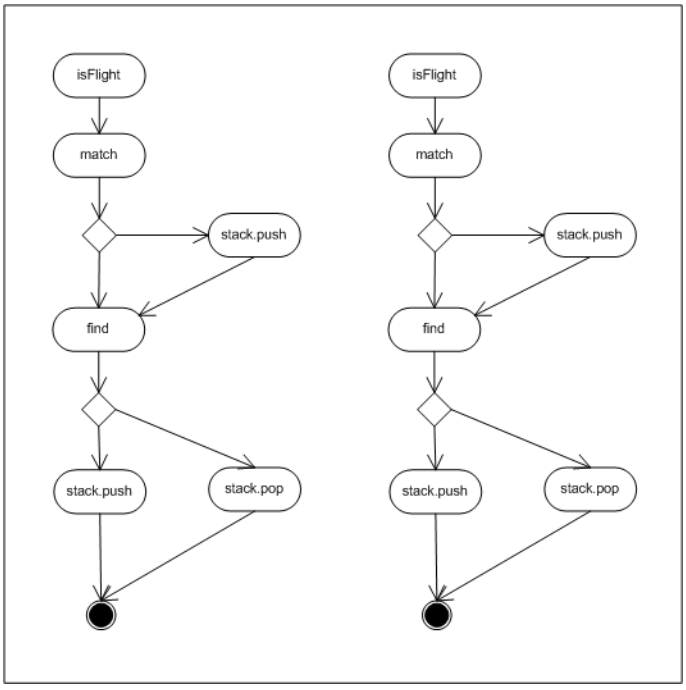
($O_1^A = \langle A_1 \| (A_2, A_3, A_4), \ ..., \ A_n \rangle$
$O_2^A = \langle A_1' \| (A_2', A_3', A_4'), \ ..., \ A_n' \rangle$ &
$(A_1 = A_1' \ \& \ A_2 = A_2' \ \& \ A_3 = A_3' \ \& \ A_4 = A_4' \ \& \ A_n = A_n')$

임을 만족하면

Control Flow 동일성 $O_1^A \sim_C O_2^A$이 성립한다. ■

3.4.1. Control Flow 동일성 예제

예제 C++프로그램의 Manager 클래스와 C프로그램의 (Manager&Stack) 클래스에 대한 Activity 다이어그램을 작성해보면 <그림 6>과 같은 결과를 얻을 수 있다.



<그림 6> C++언어와 C언어의 Control Flow 동일성 비교

이 Activity 다이어그램을 O_i^A를 통해수 표현해보면 $\langle A_{isFlight}, A_{match} \triangle (A_{stack:push}), A_{find} \triangle (A_{stack:push}, A_{stack:pop}) \rangle$의 동일한 결과 값을 갖게 되고 이를 통해 두 프로그램이 Control Flow 동일성을 만족하는 것을 알 수 있다. 이와 같이 두 프로그램이 완벽히 동일한 이유는 Activity 다이어그램이 프로그램이 갖는 처리로직을 반영하기 때문에 동일한 일을 수행하는 프로그램에 대해서는 동일한 Control Flow를 갖기 때문이며 이를 통해 두 프로그램의 제어흐름에 따른 동일성은 비교할 수 있다.

4. 결론 및 향후연구

본 논문에서는 서로 다른 패러다임의 두 언어를 모델링하여 그것들이 가지는 행위적 측면의 비교를 통해 두 프로그램의 동일성을 검증하는 방법을 제시하였다.

하나의 소프트웨어를 객체들의 집합으로 정의하고 각각의 객체를 구성하는 함수와 그것들 사이의 관계를 객체의 상태와 처리흐름으로 정의하여 이들을 이용해 State Chart 동일성과 Data Flow, Control Flow 동일성 검증 방법을 제시하였다. State Chart 동일성은 객체가 갖게 되는 상태들의 변화에 따라 내부 상태 변화의 포함여부를 구분하여 Strong State Chart 동일성과 Weak State Chart 동일성으로 분리하였다. 객체의 상태 변화에 초점을 맞춘 State Chart 동일성과는 다르게 Data Flow, Control Flow 동일성은 객체들 혹은 객체 내부의 함수들 사이의 전이 순서에 따른 동일성을 비교하였다. Data Flow 동일성의 경우 데이터가 객체들을 따라 흐르면서 변환되는 순서에 초점을 맞춰 두 소프트웨어가 동일한 객체들로 구성되었을 경우 데이터가 변환되는 순서가 동일하면 Object Data Flow 동일성을 인정하였고, 객체를 세분화하여 내부의 함수들 사이의 전이 순서가 동일할 때의 Procedure Data Flow 동일성을 정의하였다. 마지막으로 Control Flow 동일성은 소프트웨어 코드에 존재하는 처리로직을 기반으로 Activity 다이어그램을 작성하고 병렬처리와 비교 판단에 대한 처리 시점과 결과에 따른 전이순서에 대한 동일성을 비교하였다.

향후 연구에서는 좀 더 일반적인 동일성 비교를 위해 다음과 같은 연구가 필요하다. 현재의 동일성 비교는 C언어와 C++언어 사이의 동일성 비교만을 지원하고 있다. 따라서 UML과 같은 객체 중심의 모델링과 더불어 함수형 언어에 적용 가능한 모델링 기법에 대한 연구가 필요하며 이렇게 정의된 동일성 기법을 통해 둘 혹은 그 이상의 소프트웨어들 사이의 동일성 척도를 비교할 수 있을 것이다.

참고문헌

[1] Robert S. Arnold, "Software Engineering," IEEE Computher Society Press, 1994

[2] 박성옥, 노경주, 이문근, "최적합 객체 선정을 위한 다중 객체군 추출", 정보과학회논문지(B) 제 26 권, 제 12 호 1999.12

[3] Michael Siff, Thomas Reps, "Identifying Modules Via Concept Analysis", IEEE Trans. Software Eng. 25(6): 749-768 (1999)